электронный научно-технический журнал
ИНЖЕНЕРНЫЙ ВЕСТНИК
Издатель ФГБОУ ВПО "МГТУ им. Н.Э. Баумана". Эл No. ФС77-51036. ISSN 2307-0595

Сравнение методов вызова Си-функций из языка программирования Python 3

 # 11, ноябрь 2015

Васюнин А. Н.[1,*], Борисов С. В.[1]

УДК: 004.424.32

[1]Россия, МГТУ им. Н.Э. Баумана
*alex.vasyunin.1994@yandex.ru

Введение

В настоящий момент особый интерес вызывает проблема совмещения кода на языке Python 3 и Си. Этому посвящёны статьи [1, 2], в которых описывается использование библиотеки Cython для совмещения кода на C и Python, применение SWIG для решения этой же проблемы. Но в этих публикациях не уделяется должного внимания сравнению разных методов вызова Си-функций из кода, написанного на языке программирования Python 3, между собой и с функциями, написанными исключительно на языке программирования Python 3, поэтому такое сопоставление актуально.

Одним из способов достижения совмещения кода на Си и Python 3 является реализация библиотеки функций. Исходя из принципов структурного программирования [[3]], библиотека должна содержать следующие функции:

1. Пример линейной программы:
   - Вычисление синуса угла, заданного в градусах.
2. Пример разветвляющейся программы:
   - Решение квадратного уравнения по заданным коэффициентам.
3. Пример программы, использующей цикл с условием:
   - Нахождение суммы бесконечного ряда x + x/2! + x/3! + x/4! + x/5! + … с заданной точностью.
4. Пример программы, использующей цикл с заданным числом итераций:
   - Вычисление факториала.
   - Нахождение длины самой длинной последовательности строго возрастающих чисел в векторе из действительных чисел. Если подряд идут два одинаковых числа, то считаем, что началась новая последовательность.
5. Пример программы, использующей многоступенчатый цикл:
   - Найти модуль разности между максимальным и минимальным элементом матрицы из действительных чисел.

- Вывести таблицу умножения.

6. Приме обработки строк:
   - Найти первое самое длинное слово в строке.
7. Пример работы с файлами:
   - Скопировать содержимое одного файла в интервале от заданной начальной строки до заданной конечной строки в другой файл.
8. Пример работы с подпрограммами:
   - Вычисление значения выражения, записанного в обратной польской записи. Доступные операции +, -, *, /, ^ (возведение в степень).
9. Пример рекурсии:
   - Написать рекурсивный вариант функции для вычисления n-го числа Фибоначчи. Считаем, что 0 — 0-е число Фибоначчи, 1 — 1-е число Фибоначчи, 1 — 2-е число Фибоначчи.
10. Пример работы со структурами:
    - Вычисление площади треугольника. Треугольник представлен структурой Triangle, состоящей из идентификатора id типа int, координат вершин треугольника. Каждая из координат представлена структурой Point, в которой 2 действительных числа типа double.
    - Проверка нахождения точки внутри треугольника. Треугольник представлен структурой Triangle, состоящей из идентификатора id типа int, координат вершин треугольника. Каждая из координат представлена структурой Point, в которой 2 действительных числа типа double. Точка также представлена структурой Point.

Интерес также вызывает объединение этих функций в комплексную программу. Например, нахождение индекса подстроки в строке, используя алгоритм Кнута-Морриса-Пратта, расстояния Левенштейна по алгоритму Вагнера-Фишера. Результат представлен в виде структуры SubStrLevLen, состоящей из полей:

- substr -- поле типа int, представляющее индекс подстроки в строке. Если подстрока не входит в строку, то содежит -1. Нумерация, как и в Си-строках, начинается с нуля;
- levlen -- поле типа int, содержащее расстояние Левенштейна между входными строками.

Дадим некоторые разъяснения насчёт алгоритмов, используемых в описанной выше комплексной программе.

Для реализации алгоритмов предложенной комплексной программы введём некоторые определения.

Расстояние Левенштейна (также **редакционное расстояние** или **дистанция редактирования**) между двумя строками в теории информации и компьютерной лингвистике — это минимальное количество операций вставки одного символа, удаления одного символа и замены одного символа на другой, необходимых для превращения одной строки в другую.

Пусть S_1и S_2 — две строки (длиной M и N соответственно) над некоторым алфавитом, тогда редакционное расстояние (расстояние Левенштейна) d(S_1, S_2) можно подсчитать по следующей рекуррентной формуле

$$d(S_1, S_2) = D(M, N), \tag{1}$$

где

$$D(i,j) = \begin{cases} 0, \text{если } i = 0, j = 0 \\ i, \text{если } j = 0, i > 0 \\ j, \text{если } i = 0, j > 0 \\ \min(\\ D(i, j-1) + 1, \\ D(i-1, j) + 1, \\ D(i-1, j-1) + m(S_1[i], S_2[j]) \\ ) \end{cases}, \tag{2}$$

где m(a,b) равна нулю, если a = b и единице в противном случае; min(a, b, c) возвращает наименьший из аргументов.

Здесь шаг по i символизирует удаление (D) из первой строки, по j — вставку (I) в первую строку, а шаг по обоим индексам символизирует замену символа (R) или отсутствие изменений (M).

Очевидно, справедливы следующие утверждения:

- $d(S_1, S_2) \geq \big||S_1| - |S_2|\big|$
- $d(S_1, S_2) \leq \max(|S_1|, |S_2|)$
- $d(S_1, S_2) = 0 \Leftrightarrow |S_1| = |S_2|$

Для нахождения кратчайшего расстояния необходимо вычислить матрицу D, используя вышеприведённую формулу. Её можно вычислять как по строкам, так и по столбцам.

На рис. 1 представлен псевдокод алгоритма нахождения расстояния Левенштейна при произвольных ценах замен, вставок и удалений, с нумерацией символов в строке с первого.

```
1   D(0,0) = 0
2   для всех j от 1 до N
3       D(0,j) = D(0,j-1) + цена вставки символа S2[j]
4   для всех i от 1 до M
5       D(i,0) = D(i-1,0) + цена удаления символа S1[i]
6       для всех j от 1 до N
7           D(i,j) = min(
8               D(i-1, j) + цена удаления символа S1[i],
9               D(i, j-1) +        цена вставки символа S2[j],
10              D(i-1, j-1) + цена замены символа S1[i] на символ S2[j]
11          )
12  вернуть D(M,N)
```

Рис. 1. Алгоритм нахождения расстояния Левенштейна

Для восстановления редакционного предписания требуется вычислить матрицу D, после чего идти из правого нижнего угла (M,N) в левый верхний, на каждом шаге ища минимальное из трёх значений:

- если минимально (D(i-1, j) + цена удаления символа S1[i]), добавляем удаление символа S1[i] и идём в (i-1, j)
- если минимально (D(i, j-1) + цена вставки символа S2[j]), добавляем вставку символа S2[j] и идём в (i, j-1)
- если минимально (D(i-1, j-1) + цена замены символа S1[i] на символ S2[j]), добавляем замену S1[i] на S2[j] (если они не равны; иначе ничего не добавляем), после чего идём в (i-1, j-1)

Здесь (i, j) — клетка матрицы, в которой мы находимся на данном шаге. Если минимальны два из трёх значений (или равны все три), это означает, что есть 2 или 3 равноценных редакционных предписания.

Этот алгоритм называется алгоритмом Вагнера — Фишера. Он предложен Р. Вагнером (R. A. Wagner) и М. Фишером (M. J. Fischer) в 1974 году.

Алгоритм Кнута-Морриса-Пратта основан на принципе конечного автомата. В этом алгоритме состояния помечаются символами, совпадение с которыми должно в данный момент произойти. Из каждого состояния имеется два перехода: один соответствует успешному сравнению, другой — несовпадению. Успешное сравнение переводит нас в следующий узел автомата, а в случае несовпадения мы попадаем в предыдущий узел, отвечающий образцу. Автомат Кнута-Морриса-Пратта для ababcb изображён на рис. 2.

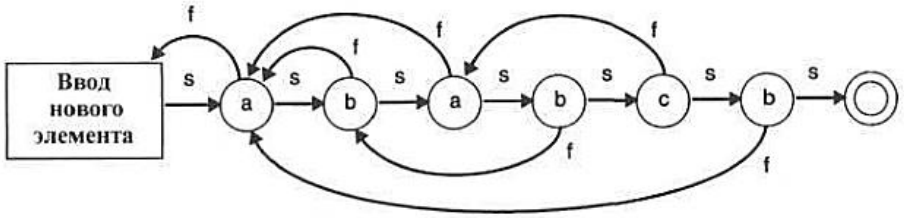


Рис. 2. Автомат Кнут-Морриса-Пратта для ababcb

При всяком переходе по успешному сравнению в конечном автомате Кнута-Морриса-Пратта происходит выборка нового символа из текста. Переходы, отвечающие неудачному сравнению, не приводят к выборке нового символа; вместо этого они повторно используют последний выбранный символ. Если мы перешли в конечное состояние, то это означает, что искомая подстрока найдена. Вот псевдокод полного алгоритма поиска показан на рис. 3.

```
1   subLoc=l // указатель тек. сравниваемого символа в подстроке
2   textLoc=l // указатель тек. сравниваемого символа в тексте
3   while textLoc<=length(text) and subLoc<=length(substring) do
4           if subLoc=0 or text[textLoc]=substring[subLoc] then
5                   textLoc=textLoc+l
6                   subLoc=subLoc+l
7           else // совпадения нет; переход по несовпадению
8                   subLoc=fail[subLoc]
9   if (subLoc>length(substring) then
10          return textLoc-length(substring)+l // наиденное совпадение
11  else
12          return 0 // искомая подстрока не найдена
```

Рис. 3. Алгоритм поиска подстроки

Прежде, чем перейти к анализу этого процесса, рассмотрим, как задаются переходы по несовпадению. Заметим, что при совпадении ничего особенного делать не надо: происходит переход к следующему узлу. Напротив, переходы по несовпадению определяются тем, как искомая подстрока соотносится сама с собой. Например, при поиске подстроки ababcb нет необходимости возвращаться назад на четыре позиции при необнаружении символа c. Если уж мы добрались до пятого символа в образце, то мы знаем, что первые четыре символа совпали, и поэтому символы ab в тексте, отвечающие третьему и четвертому символам образца, совпадают также с первым и вторым символами образца. На рис. 4 показан псевдокод алгоритма, задающего эти соотношения в подстроке.

```
1       fail[l]=0
2       for i=2 to length(substring) do
3               temp=fail[i-l]
4               while(temp>0) and substring[temp]/=substring[i-1]) do
5                       temp=fail[temp]
6               fail[i]=temp+l
```

Рис. 4. Построение конечного автомата

Для реализации сравнения был создан программный продукт, обладающий графическим интерфейсом для наглядного представления результатов исследований. Структура его модулей показана на рис. 5.

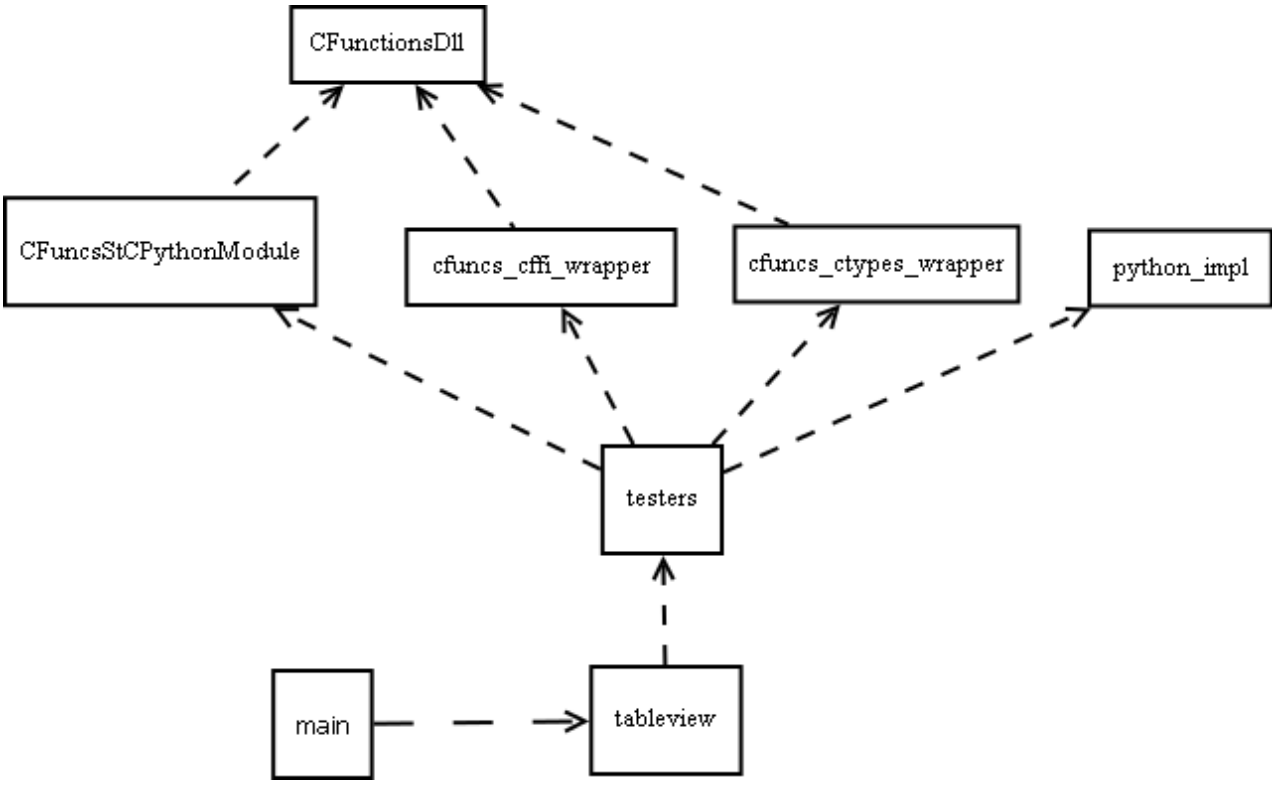


Рис. 5. Структура модулей

Дадим краткое описание способа вызова Си-функций из кода на языке Python без использования сторонних библиотек. В файле cfuncs_stcpython.cpp содержится обёртка, при помощи которой Си-функции будут вызываться из кода Python. Его код представлен на рис.6.

```
1   #include "include.h"
2   #include "cfunctions.h"
3   // обёртка sin_degrees
4   static PyObject *cfuncs_stcpython_sin_degrees(PyObject *self, PyObject *args) {
5           double x;
6           if(!PyArg_ParseTuple(args, "d", &x))
7                   return NULL;
8           else {
9                   unsigned int memory = 0;
10                  double result = sin_degrees(x, &memory);
11                  return Py_BuildValue("(dI)", result, memory);
12          }
13  }
14  // имена функций
15  static PyMethodDef SpamMethods[] = {
16          {"sin_degrees", cfuncs_stcpython_sin_degrees, METH_VARARGS, "sin(x)"},
17          {NULL, NULL, 0, NULL}
18  };
19  // свойства модуля
20  static struct PyModuleDef cfuncs_stcpython = {
21          PyModuleDef_HEAD_INIT,
22          "cfuncs_stcpython", // имя модуля
23          NULL, // указатель на документацию
24          -1,
25          SpamMethods
26  };
27  // инициализация модуля
28  PyMODINIT_FUNC PyInit_cfuncs_stcpython(void) {
29          PyObject *m;
30          m = PyModule_Create(&cfuncs_stcpython);
31          if (m == NULL)
32                  return NULL;
33          return m;
34  }
```

Рис. 6. Пример кода модуля

Здесь cfuncs_stcpython_sin_degrees — непосредственная обёртка функции sin_degrees. Подобные функции будут написаны для всех функций, которые мы хотим импортировать в Python. Все эти функции-обёртки возвращают указатель на PyObject и принимают два указателя на PyObject. Также эти функции объявлены как static. Все эти функции-обёртки нужно зарегистрировать в static PyMethodDef SpamMethods[]. Это массив структур PyMethodDef. PyMethodDef содержит имя функции в Python, имя-функции-обёртки, свойства передачи аргументов и краткое пояснение, что делает эта функция. Этот массив структур нужен, чтобы Python знал, какие функции-обёртки вызывать при написании имени необходимой нам функции.

static struct PyModuleDef cfuncs_stcpython — здесь устанавливаем свойства модуля. Здесь устанавливаем указатель на массив структур PyMethodDef.

PyMODINIT_FUNC PyInit_cfuncs_stcpython(void) — функция для инициализации модуля. Здесь мы указатель на структуру PyModuleDef, чтобы Python смог впоследствии найти функции-обёртки.

В функции-обёртке мы сначала разбираем входные аргументы при помощи PyArg_ParseTuple. Первый аргумент этой функции – это аргументы из кода Python, потом идёт строка формата, а затем указатели из кода на Си, куда мы будем записывать разобранные аргументы. Если аргументы разобраны удачно, то функция PyArg_ParseTuple возвращает ненулевое значение. Если вернуть NULL из функции-обёртки, то Python воспримет это как то, что в Си-функции возникла ошибка.

Затем в функции-обёртке вызывается непосредственно Си-функция. Возвращаем значения в Python при помощи функции Py_BuildValue. В Py_BuildValue сначала идёт строка формата возвращаемых аргументов, а потом сами аргументы.

Аналогичные функции обёртки нужно написать для каждой функции, которую мы хотим вызвать из кода Python, а потом добавить указатели на функции-обёртки в массив структур PyMethodDef.

В коде Python нужно импортировать созданный нами файл .pyd. Для этого пишем import и путь к файлу .pyd без этого расширения. Теперь функцию sin_degrees можно вызвать из кода Python как обычную функцию, написанную на этом языке.

Дадим краткое описание способа вызова Си-функций из кода на языке Python с использованием библиотеки cffi.

Создадим файл cfuncs_cffi_wrapper.py. Этот файл будет содержать Python-обёртки для вызова Си-функций. Затем импортируем модуль cffi и создадим объект ffi класса FFI для загрузки Си-функций из написанной нами библиотеки. Запишем сигнатуры функций, которые мы будем вызывать в Python-коде. Они показаны на рис. 7.

```
1   ffi.cdef("""
2   // линейные программы
3   double normal_distr_density(double x, double mu, double sigma, unsigned int *memory);
4   double squared_x(double x, unsigned int *memory);
5   double sin_degrees(double degrees, unsigned int *memory);
6   int func(int x);
7   // программы с ветвлением
8   const char *solve_square_equation(double a, double b, double c, unsigned int *memory);
9   // цикл с условием
10  double sum_of_infinite_series(double x, double epsilon, unsigned int *memory);
11  // цикл с заданным числом итераций
12  double factorial(unsigned int number, unsigned int *memory);
13  unsigned int get_max_len_incr_series(double *arr, unsigned int count, unsigned int *memory);
14  // вложенные циклы
15  double get_abs_dif_max_min(double *matr, int nrows, int ncolumns, unsigned int *memory);
16  void multipl_table(unsigned int *memory);
17  // работа со строками
18  const char *get_first_longest_word(char *str, unsigned int *memory);
19  // файлы
20  const int BUF_SIZE = 1001;
21  typedef enum {ERROR_OPEN_READ=1, ERROR_OPEN_WRITE} file_errors;
22  int copy_strs(const char *input, const char *output, unsigned int start, unsigned int end);
23  // подпрограммы
24  const int MAX_D = 256;
25  double compute_rpn(char *str, unsigned int *memory);
26  // рекурсия
27  int compute_fib(int number);
28  """)
```

Рис. 7. Сигнатуры функций

Далее в переменной path укажем в виде строки путь, где находится библиотека функций на Си. После этого непосредственно загрузим написанную библиотеку вызовом метода ffi.dlopen(path) и присваеванием возвращаемого им значения переменной cffi_lib. Теперь мы можем вызвать из Python-кода любую функцию, которую указали в ffi.cdef. Однако для удобства вызова нужно создать функцию-обёртку на Python. Рассмотрим пример создания обёртки на примере get_abs_dif_max_min, изображённый на рис. 8.

```
1   def get_abs_dif_max_min(matr, nrows, ncolumns):
2           memory = ffi.new("unsigned int *")
3           addr, size = matr.buffer_info()
4           double_arr = ffi.cast("double *", addr)
5           result = cffi_lib.get_abs_dif_max_min(double_arr, nrows, ncolumns, memory)
6           return (result, memory[0])
```

Рис. 8. Функция-обёртка в cffi

При помощи вызова ffi.new(“unsigned int *”) создаётся указатель на unsigned int. Вызывая matr.buffer_info() я получаю адрес нулевого элемента матрицы matr. При помощи

ffi.cast происходит преобразование типа void * к double *. После этого cffi_lib.get_abs_dif_max_min вызывает Си-функцию и передаёт туда преобразованные необходимым образом данные.

При работе с cffi следует обратить внимание на передачу массивов в функции, передачу строк, указателей. Если в качестве типа символа строки в Си-функции используется char, то строки в коде на Python, нужно сначала преобразовать в байты, а затем передавать байты в функцию. Если мы возвращаем char * из Си-функции, то такую строку аналогично нужно преобразовать из байтов в Python-строку.

Дадим краткое описание способа вызова Си-функций из кода на языке Python с использованием библиотеки ctypes. Создадим файл cfuncs_ctypes_wrapper.py. Этот файл будет содержать Python-обёртки для вызова Си-функций. Для начала импортируем модуль ctypes, затем присвоим переменной lib_path путь к написанной библиотеке Си-функций. Вызовом метода cdll.LoadLibrary(lib_path) и присвоением возвращаемого им значения переменной ctypes_lib осуществляется загрузка библиотеки Си-функций. Теперь можно вызвать любую функцию из CfunctionsDll.dll, но для удобства напишем обёртки. Разберём написание обёртки на примере функции sin_degrees, исходный код которой показан на рис. 9.

```
1   def sin_degrees(degrees):
2           f = ctypes_lib.sin_degrees
3           f.restype = c_double
4           memory = c_int(0)
5           p_memory = pointer(memory)
6           degrs = c_double(degrees)
7           result = f(degrs, p_memory)
8           return (result, memory.value)
```

Рис. 9. Функция-обёртка в ctypes

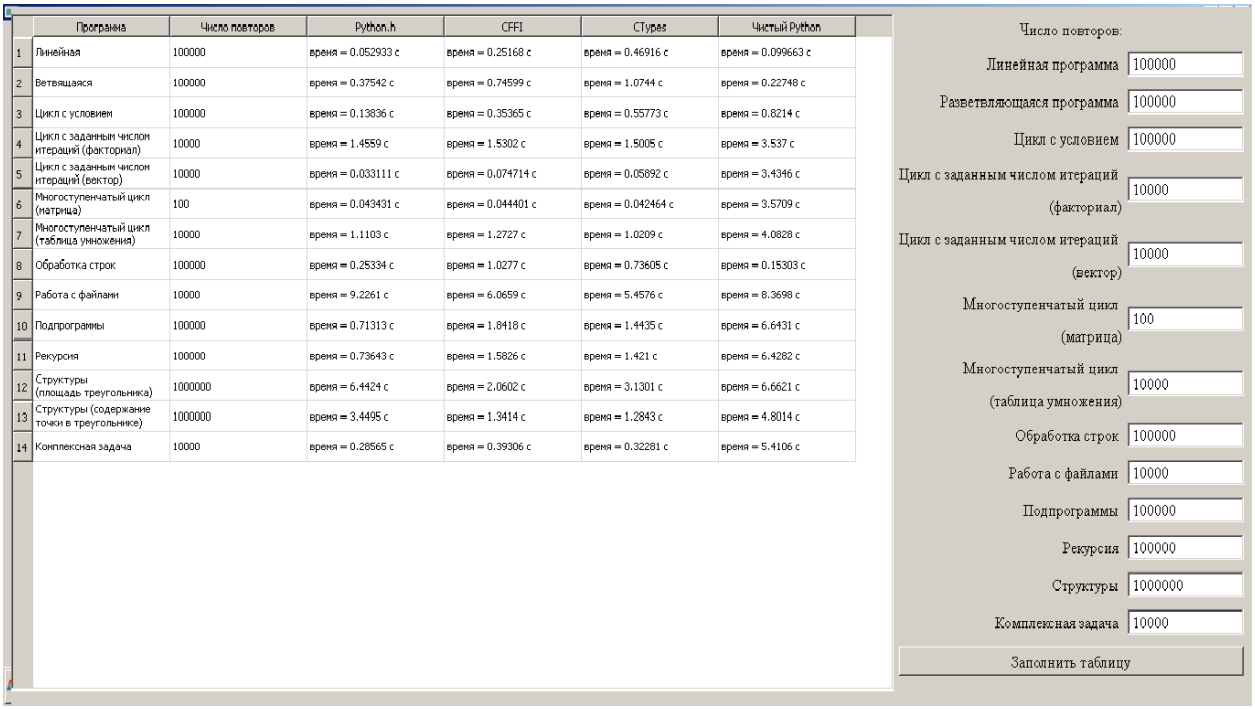
По умолчанию в библиотеке ctypes принято, что функция возвращает значение типа int. Если это не так, необходимый нам тип нужно указывать явно. В данном примере это делается при вызове f.restype = c_double.

Указатель создаётся вызовом функции pointer. Далее надо преобразовать типы входных параметров в соответствующие им Си-типы. В данном примере это делается вызовом degrs = c_double(degrees). Затем идёт непосредственный вызов Си-функции. В неё мы передаём аргументы с преобразованными типами к Си-типам.

При передаче строк в Си-функцию, где они представлены как char *, Python-строки следует привети к массиву байтов. При возврате char * из Си-функции надо массив байтов преобразовать к Python-строкам

Обёртки остальных Си-функций реализуются аналогично.

Функции, решающие те же самые задачи, что и подпрограммы, написанные на Си, были реализованы на чистом Python 3 и помещены в модуль python_impl. Это было сделано для сравнения быстродействия Си-функций и Python-функций.



	Программа	Число повторов	Python.h	CFFI	CTypes	Чистый Python
1	Линейная	100000	время = 0.052933 с	время = 0.25168 с	время = 0.46916 с	время = 0.099663 с
2	Ветвящаяся	100000	время = 0.37542 с	время = 0.74599 с	время = 1.0744 с	время = 0.22748 с
3	Цикл с условием	100000	время = 0.13836 с	время = 0.35365 с	время = 0.55773 с	время = 0.8214 с
4	Цикл с заданным числом итераций (факториал)	10000	время = 1.4559 с	время = 1.5302 с	время = 1.5005 с	время = 3.537 с
5	Цикл с заданным числом итераций (вектор)	10000	время = 0.033111 с	время = 0.074714 с	время = 0.05892 с	время = 3.4346 с
6	Многоступенчатый цикл (матрица)	100	время = 0.043431 с	время = 0.044401 с	время = 0.042464 с	время = 3.5709 с
7	Многоступенчатый цикл (таблица умножения)	10000	время = 1.1103 с	время = 1.2727 с	время = 1.0209 с	время = 4.0828 с
8	Обработка строк	100000	время = 0.25334 с	время = 1.0277 с	время = 0.73605 с	время = 0.15303 с
9	Работа с файлами	10000	время = 9.2261 с	время = 6.0659 с	время = 5.4576 с	время = 8.3698 с
10	Подпрограммы	100000	время = 0.71313 с	время = 1.8418 с	время = 1.4435 с	время = 6.6431 с
11	Рекурсия	100000	время = 0.73643 с	время = 1.5826 с	время = 1.421 с	время = 6.4282 с
12	Структуры (площадь треугольника)	1000000	время = 6.4424 с	время = 2.0602 с	время = 3.1301 с	время = 6.6621 с
13	Структуры (содержание точки в треугольнике)	1000000	время = 3.4495 с	время = 1.3414 с	время = 1.2843 с	время = 4.8014 с
14	Комплексная задача	10000	время = 0.28565 с	время = 0.39306 с	время = 0.32281 с	время = 5.4106 с

Рис. 10. Результаты тестов

На рис. 10 показаны результаты работы программного комплекса. На основе полученной таблицы можно сделать выводы:

1. В большинстве тестов выигрывает стандартный метод, использующий подключение файла Python.h, который требует писать обёртку в Си.
2. В тестах на обработку строк выиграл чистый Python. Значит, обработку строк можно с хорошим быстродействием производить на чистом Python, не используя функции на Си, используя встроенную библиотеку Python для работы со строками.
3. Чистый Python не очень хорошо себя показал в задачах с относительно интенсивными математическими вычислениями.
4. Чистый Python проиграл значительно в обработке массивов. Это связано с неэффективной организацией доступа к элементам массива, то есть неэффективной реализацией индексации.
5. Методы Ctypes и Cffi идут примерно вровень. На некоторых тестах Ctypes обыгрывает Cffi, на других Cffi обыгрывает Ctypes.
6. Стандартный метод, использующий подключение файла Python.h, который требует писать обёртку в Си, лидирует в тестах, которые связаны с относительно интенсивными математическими вычислениями.
7. При работе со структурами методы Cffi и Ctypes показали меньшее время.
8. Самым трудоёмким по реализации является метод, использующий подключение файла Python.h, который требует писать обёртку в Си, методы Ctypes и Cffi примерно равны по трудоёмкости. Если выбирать универсальный метод вызова Си-функций из Python, то, оценив простоту реализации и производительность, я выберу Cffi.

Заключение

Итак, была написана библиотека функций на Си, программа на языке Python, в которой тремя разными способами были вызваны эти функции. В программе было осуществлено сравнение этих методов между собой и с реализацией на чистом Python. Результаты представлены в виде таблицы, созданной с использование библиотеки PyQt5.

Список литературы

[1]. Behnel S., Bradshaw R., Citro C., Dalcin L., Seljebotn D.S., Cython K. S. The Best of Both Worlds. // Computing in Science & Engineering. Publisher: IEEE Xplore. 2011. Vol. 13. Is. 2. P. 31-39. DOI: 10.1109/MCSE.2010.118

[2]. Cottom T.L. Using SWIG to Bind C++ to Python. // Computing in Science & Engineering. Publisher: IEEE Xplore. 2003. Vol. 5. Is. 2. P. 88 - 97. DOI: 10.1109/MCISE.2003.1182968

[3]. Иванова Г.С. Программирование: учебник для вузов. 3-е изд., стер. М.: Кнорус. 2014. 425 с.

[4]. Документация по языку Python. // Python: официальный сайт. Режим доступа: https://www.python.org/ (дата обращения: 14.09.2015).

[5]. Extending and Embedding the Python Interpreter. / Расширение интерпретатора Python при помощи расширений на Си. // Python: официальный сайт. Режим доступа: https://docs.python.org/3/extending/index.html (дата обращения: 14.09.2015).

[6]. CFFI documentation / Документация по библиотеке Cffi // CFFI: официальный сайт. Режим доступа: http://cffi.readthedocs.org/en/latest/ (дата обращения: 14.09.2015).

[7]. 16.16. ctypes — A foreign function library for Python / Документация по библиотеке Ctypes. // Python: официальный сайт. Режим доступа: https://docs.python.org/3/library/ctypes.html (дата обращения: 14.09.2015).

[8]. Summerfield M. Programming in Python 3: A Complete Introduction to the Python Language. 2nd ed. Addison-Wesley Professional. 2009. 648 p.

[9]. Керниган Б.У., Ритчи Д.М. Язык программирования Си. 3-е изд., испр. М.: Вильямс. 2006. 304 с.